# A generalized approach for Boolean matrix factorization

Rodrigo Cabral Farias[a,*], Sebastian Miron[b]

[a] *Université Côte d'Azur, CNRS, I3S Laboratory, 06900 Sophia-Antipolis, France*
[b] *Université de Lorraine, CNRS, CRAN, 54000 Nancy, France*

**Abstract**

In this paper, we propose a generalized framework for fitting Boolean matrix factorization models to binary data. In this generalized setting, the binary rank-1 components of the underlying model can be combined by any Boolean function, thus extending the standard Boolean matrix factorization model, where the combination is restricted to the logical 'OR' function. We introduce two algorithms relying on a relaxation of the binary constraints on the factors of the model and on a polynomial representation of the Boolean function that combines the rank-1 components. One of the algorithms is based on the gradient descent optimization method, while the other is based on block coordinate descent. A detailed presentation of the algorithms is given, along with numerical experiments both on simulated and real datasets. A comparison with other algorithms from the literature is presented in the standard Boolean matrix setting allowing to assess the advantages and shortcomes of the proposed methods in terms of factor retrieval and data denoising performance, convergence behavior and time complexity.

*Keywords:* Binary data, Binary matrix factorizations, Boolean matrix factorizations, Data mining.

## 1. Introduction

Binary data matrices are one of the most natural ways of numerically encoding data in many applications. They can encode *yes*/*no* answers to surveys, voting records, tables indicating the presence or absence of traits of a group of individuals or indicating the proximity between the individuals, implicit feedback or thresholded explicit feedback (grades) in audio/video streaming platforms, input/output relationships in digital circuits, among many others. For this reason, a large number of data mining techniques specially tailored for binary data matrices have been developed

*Corresponding author
*Email address:* Rodrigo.CABRAL-FARIAS@univ-cotedazur.fr (Rodrigo Cabral Farias)
*URL:* https://www.i3s.unice.fr/rcabral/en/home (Rodrigo Cabral Farias)

recently. Among these techniques, a diverse number of models and algorithms relying on matrix factorizations have appeared in the last two decades. They have been successfully used in a great number of applications such as text mining [1], recommender systems [2–4], genetics [5, 6], protein complex prediction [7], role mining [8] and telecommunications [9, 10].

Different factorization models for binary matrices have been proposed in the literature. One group of models [2, 11–14] relies on a modification of logistic regression where a logistic function is applied to a constrained matrix factorization model such as principal component analysis (PCA). Most models of this group are named logistic PCA or binary PCA. Although the matrix factors are not constrained to be binary, the use of the logistic function allows the model to constrain the elements of its output matrix to lie in the interval $[0, 1]$. A generalized version of logistic PCA, based on the family of mean-parameterized Bernoulli models, was recently studied in [15].

Another model, called binary matrix factorization (bMF) [16], directly factorizes the data matrix into two binary matrices. This model is equivalent to a decomposition of the data matrix into a sum of rank-1 binary matrices. A major issue for fitting optimally a bMF model to data is the discrete nature of the model parameters, which makes the underlying optimization problem difficult to be solved. It has been shown that fitting a single component bMF model is already a NP-hard problem [17]. Due to its hardness, algorithms for bMF do not aim to solve exactly the original fitting problem. A class of methods, such as the association rules algorithm (ASSO) [18] or formal concept analysis (FC) [19], rely on low complexity iterative rules that extract in a greedy-like manner binary rank-1 components that are expected to approximate the optimal ones. A revisited version of the FC-based algorithms of [19], significantly faster, was recently introduced in [20]; another variant, that uses the minimum description length principle (MDL) for factor selection was proposed in [21]. A different approach, called the penalty function algorithm (PF) [16], solves a relaxed version of the underlying fitting problem. In PF, the bMF factor elements are relaxed to the nonnegative orthant, while a penalization term forcing the factors to be close to binary is added to the original data fitting objective function. Such a relaxation allows to use multiplicative gradient algorithms to retrieve the factors in a similar manner as for nonnegative matrix factorization (NMF) [22].

The limitations of bMF appear whenever its expected rank-1 components have overlapping supports. In this case, the sum of the rank-1 components does not lead to a binary matrix. One way to counter this issue is to assume that the presence of a '1' in the data matrix is due to the contributions of several '1' in the rank-1 terms. Note that this corresponds to simply replacing the arithmetic sums in the decomposition model by logical 'OR' operations. This leads to a particular

matrix factorization model, called Boolean matrix factorization (BMF).

A modification of the PF algorithm explicitly tailored for BMF has been proposed in [23]. The authors propose to apply a threshold function to the bMF model, such that the output matrix elements are either 0 or 1. Since the threshold function is not differentiable, to be able to use a multiplicative gradient algorithm as in the PF algorithm, a smooth approximation of the threshold function is used. The resulting BMF method is called post-nonlinear PF algorithm (PNL-PF). A heuristic model selection algorithm for estimating the number of binary sources in the BMF setting was also proposed in [24], based on stability criteria. This method constructs an ensemble of random matrices that are slight perturbations of the initial matrix to test the stability of the Boolean decomposition. Other algorithms have also been developed under a stochastic setting, where the elements of the factors are supposed to be random [25, 26].

In this paper, we propose an approximate factorization approach for binary valued matrices that generalizes BMF to arbitrary Boolean "*sum*" functions. Instead of considering combinations of the rank-1 components with logical 'OR', we assume that an arbitrary Boolean function with known truth table is used. Our approach is based on the relaxation of the binary constraints, as in PF and PNL-PF, but instead of representing the behavior of the logical combiner with a threshold function, we represent it as a multivariate polynomial of the elements of each component. Since a multivariate polynomial is a differentiable function, such a representation allows developing a gradient algorithm for fitting the generalized BMF model, without the need to resort to smooth approximations, as in PNL-PF. We also propose a generalized BMF approximation algorithm based on block coordinate descent. The algorithm alternatively updates the columns of the factors to be retrieved in a similar manner as in hierarchical alternating least squares (HALS) for NMF [27]. Since the multivariate polynomial representing the logical combiner is multilinear in the elements of the components, the updates required in the block coordinate descent algorithm can be obtained in closed-form.

We present implementation details of our approach in the specific case of of BMF approximation, and under this setting we compare the performance of the two resulting algorithms with state-of-the art BMF methods. The performance of the methods are evaluated in terms of denoising and factor retrieval capabilities, but also in terms of convergence behavior, time complexity and sensitivity to initializations. To illustrate our approach in a more practical setting, we apply one of the proposed methods to retrieve the BMF of 4 real datasets. Finally, we also show simulation results concerning the application of the general version of the proposed algorithms to retrieve factorizations where the component combining functions are the logical exclusive 'OR' ('XOR') and the 3-term majority

function.

### 1.1. Outline

In Section 2, we present the binary and Boolean factorization models along with the polynomial representation of the general Boolean factorization. We introduce two algorithms for the generalized Boolean factorization in Section 3 and illustrate their implementation in the specific case of BMF. Section 4 shows the results of the conducted numerical experiments to compare the performance of the proposed algorithms with state-of-the art methods. Results on 4 real datasets are also given. We also provide numerical simulation results in a more general setting, where the data do not follow the standard BMF model. Finally, we conclude the paper in Section 5.

### 1.2. Notations

Scalars are represented by lower case letters $x$, while vectors are represented by bold-face lower case letters $\boldsymbol{x}$. Bold-face upper case letters $\boldsymbol{X}$ are used to represent matrices. A single subscript $x_i$ is used to represent the $i$-th element of a vector or the $i$-th column of a matrix $\boldsymbol{x}_i$ ($i$-th column of $\boldsymbol{X}$). A double subscript $x_{ij}$ denotes the $(i,j)$-th element of a matrix. Superscripts (or subscripts) of the form $\boldsymbol{x}^{1:n}$ denote the tupple $(\boldsymbol{x}^1, \boldsymbol{x}^2, \cdots, \boldsymbol{x}^n)$.

Matrix transpose is denoted $\boldsymbol{X}^\mathsf{T}$, while the Frobenius norm of a matrix is symbolized by $\|\boldsymbol{X}\|_F$. To denote a matrix of size $I \times J$ with all elements equal to 1, we use $\mathbf{1}_{I\times J}$. The symbol $\boxdot$ denotes the Hadamard (entry-wise) matrix product and $\text{Diag}(\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_n)$ denotes a block diagonal matrix with matrices $\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_n$ in its diagonal blocks. The column-major vectorization of a matrix is denoted $\text{vec}(\boldsymbol{X})$.

Logical 'OR' is symbolized by $\vee$ and the same symbol is used for its entry-wise matrix version. Logical 'XOR' is denoted by $\oplus$.

## 2. General binary and Boolean factorizations

We are interested in exactly or approximately decomposing a $I \times J$ data matrix $\boldsymbol{Y}$ with binary elements $y_{ij} \in \{0, 1\}$, for $(i,j) \in \{1, 2, \cdots, I\} \times \{1, 2, \cdots, J\}$, into $R \geqslant 2$ binary rank-1 matrices $\boldsymbol{X}^{1:R} = \left\{\boldsymbol{X}^1, \boldsymbol{X}^2, \cdots, \boldsymbol{X}^R\right\}$. Each binary $\boldsymbol{X}^r$ with $r \in \{1, 2, \cdots, R\}$ is written as

$$\boldsymbol{X}^r = \boldsymbol{a}_r \boldsymbol{b}_r^\mathsf{T}, \tag{1}$$

where $\boldsymbol{a}_r$ and $\boldsymbol{b}_r$ are vectors of sizes $I$ and $J$ respectively and with their elements constrained to be binary $[\boldsymbol{a}_r]_i \in \{0, 1\}$, for $(i,r) \in \{1, 2, \cdots, I\} \times \{1, 2, \cdots, R\}$, $[\boldsymbol{b}_r]_j \in \{0, 1\}$, for $(j,r) \in$

$\{1, 2, \cdots, J\} \times \{1, 2, \cdots, R\}$. The vectors $\boldsymbol{a}_r$ and $\boldsymbol{b}_r$ can be stored in matrices $\boldsymbol{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_R]$ and $\boldsymbol{B} = [\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_R]$. As presented in [23] and [28], different decompositions can be considered depending on how one precisely defines the way that $\boldsymbol{X}_r$ are combined to approximate $\boldsymbol{Y}$. If we specify a function $f : \{0, 1\}^R \to \mathcal{Y} \subset \mathbb{R}$ defined on $R$ binary inputs and resulting in a value on a finite subset $\mathcal{R}$ of the integers, general binary factorization corresponds to approximate the elements of $\boldsymbol{Y}$ as follows:

$$y_{ij} \approx f\left(x_{ij}^1, x_{ij}^2, \cdots, x_{ij}^R\right) = f\left(x_{ij}^{1:R}\right) = f\left(a_{i,1:R} b_{j,1:R}\right), \tag{2}$$

where $a_{i,1:R} b_{j,1:R} = \{a_{i1} b_{j1}, a_{i2} b_{j2}, \cdots, a_{iR} b_{jR}\}$. Denoting the matrix resulting of the element-wise application of $f(\cdot)$ to the rank-one matrices $\boldsymbol{X}^{1:R}$ simply by $f(\boldsymbol{X}^{1:R})$, the approximation problem (2) can be cast as the following minimization problem:

$$\begin{aligned} \text{minimize} \quad & \mathcal{F}(\boldsymbol{A}, \boldsymbol{B}) = \frac{1}{2} \|\boldsymbol{Y} - f(\boldsymbol{X}^{1:R})\|_F^2 \\ (\text{where} \quad & \boldsymbol{X}^{1:R} = \left\{\boldsymbol{X}^1 = \boldsymbol{a}_1 \boldsymbol{b}_1^\mathsf{T}, \cdots, \boldsymbol{X}^R = \boldsymbol{a}_R \boldsymbol{b}_R^\mathsf{T}\right\}), \\ \text{with respect to} \quad & \boldsymbol{A} \in \{0, 1\}^{I \times R}, \boldsymbol{B} \in \{0, 1\}^{J \times R}. \end{aligned} \tag{3}$$

A solution for this problem is guaranteed to exist since the cardinal of the feasible set is finite. If a solution $\boldsymbol{A}^\star$, $\boldsymbol{B}^\star$ of (3) achieves $\mathcal{F}(\boldsymbol{A}, \boldsymbol{B}) = 0$, we say that it is an exact factorization of $\boldsymbol{Y}$. In the data analysis literature, 3 common types of factorization problems which are special cases of the general form above are the following:

*Binary matrix factorization (bMF).* When $f(x_{ij}^{1:R}) = \sum_{r=1}^R x_{ij}^{1:R}$ is the usual sum on $\mathbb{R}$. See the left column of Tab. 1 for an example when $R = 2$. We say that $\boldsymbol{A}$ and $\boldsymbol{B}$ are approximate factors of $\boldsymbol{Y}$, since in this case $\boldsymbol{Y} \approx \boldsymbol{A}\boldsymbol{B}^\mathsf{T}$.

*Boolean matrix factorization (BMF).* When $f(x_{ij}^{1:R}) = \bigvee_{r=1}^R x_{ij}^{1:R}$ is the $R$-term logical 'OR'. See the middle column of Tab. 1 for an example. Similarly to the previous case, we can say that $\boldsymbol{A}$ and $\boldsymbol{B}$ are approximate Boolean factors of $\boldsymbol{Y}$, since $\boldsymbol{Y} \approx \boldsymbol{A} \wedge \boldsymbol{B}^\mathsf{T}$ where $(\cdot \wedge \cdot)$ is the matrix product defined in the Boolean semi-ring (sums are replaced by logical 'OR').

$\mathbb{F}_2$ *matrix factorization (F2MF).* When $f(x_{ij}^{1:R}) = \bigoplus_{r=1}^R x_{ij}^{1:R}$ is the $R$-term modulo-2 sum, that is, a cascade of $R$ logical 'XOR' operations applied sequentially to $x_{ij}^{1:R}$. In this case, we can write $\boldsymbol{Y} \approx \boldsymbol{A} \odot \boldsymbol{B}^\mathsf{T}$, where $(\cdot \odot \cdot)$ is the matrix product in the $\mathbb{F}_2$ field (Galois field of two elements). Here the sums are replaced by logical 'XOR'. Therefore, we call this model $\mathbb{F}_2$ matrix factorization.

Observe that if one wants to factorize a binary data matrix $\boldsymbol{Y}$ without errors using bMF, its factors should contain columns with disjoint supports. This is due to the presence of possible values larger than 1 in the outputs of the sum operation for bMF (see Tab. 1).

| Inputs | | Output | | |
|---|---|---|---|---|
| $x^{(1)}$, | $x^{(2)}$ | bMF ($+$) | BMF ($\vee$) | F2MF ($\oplus$) |
| 0, | 0 | 0 | 0 | 0 |
| 0, | 1 | 1 | 1 | 1 |
| 1, | 0 | 1 | 1 | 1 |
| 1, | 1 | 2 | 1 | 0 |

Table 1: Results for different $f(\cdot)$ with $R = 2$ inputs used in different factorizations.

For a given $\boldsymbol{Y}$, the characteristics of its factorization such as rank or uniqueness may change depending on the chosen $f(\cdot)$. Before focusing on algorithms for general Boolean factorizations, which are the main contribution of this paper, we briefly illustrate with some toy examples, some important differences between factorizations with different $f(\cdot)$.

### *2.1. Ranks and uniqueness of binary factorizations*

As in standard matrix factorizations, the minimal number of columns $R$ for which exact factorizations of $\boldsymbol{Y}$ exist is called the rank of $\boldsymbol{Y}$ and we denote it $\text{rank}_f(\boldsymbol{Y})$:

$$\text{rank}_f(\boldsymbol{Y}) = \min \left\{ R \,|\, \boldsymbol{Y} = f\left(\boldsymbol{X}^{1:R}\right), \boldsymbol{A} \in \{0,\,1\}^{I\times R}, \boldsymbol{B} \in \{0,\,1\}^{J\times R} \right\}. \tag{4}$$

Following the denominations in [23, 28], if $f(\cdot)$ is the standard sum, we call this rank the *binary rank* and we denote it $\text{rank}_{\{0,1\}}(\boldsymbol{Y})$. If $f(\cdot)$ is the logical 'OR' then this rank is the *Boolean rank* and it is denoted $\text{rank}_{\mathbb{B}}(\boldsymbol{Y})$. We call it $\mathbb{F}_2$ *rank*, when the combining function is the modulo-2 sum and we denote it $\text{rank}_{\mathbb{F}_2}(\boldsymbol{Y})$. In what follows, we give some toy examples from the literature to illustrate the fact that these ranks can be different for a given matrix. Consider the $3 \times 3$ binary matrix [18]

$$\boldsymbol{Y}_1 = \begin{bmatrix} 1\,1\,0 \\ 1\,1\,1 \\ 0\,1\,1 \end{bmatrix}. \tag{5}$$

Minimum rank decompositions of $\boldsymbol{Y}_1$ for bMF and BMF are

$$\boldsymbol{Y}_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}^\mathsf{T} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}^\mathsf{T} + \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^\mathsf{T} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}^\mathsf{T} \vee \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}^\mathsf{T}$$

and the same factors of bMF can be used for F2MF. Thus, we have $\text{rank}_{\mathbb{F}_2}(\boldsymbol{Y}_1) = 3$, which is larger than $\text{rank}_{\mathbb{B}}(\boldsymbol{Y}_1) = 2$. Note also that the rank of $\boldsymbol{Y}_1$ on the reals is $\text{rank}(\boldsymbol{Y}_1) = 3$, since the 3 columns of $\boldsymbol{Y}_1$ are linearly independent.

One can easily find cases where the relations between these ranks are different from the previous example. Consider a matrix $\boldsymbol{Y}_2$ which is equal to $\boldsymbol{Y}_1$ except for the central element $[Y_1]_{2,2}$ which is

flipped to zero. Then the BMF factors for $\boldsymbol{Y}_1$ give an exact F2MF for $\boldsymbol{Y}_2$ with a minimum number of columns. In this case, $\text{rank}_{\mathbb{B}}(\boldsymbol{Y}_2) = \text{rank}(\boldsymbol{Y}_2) = 3$, but $\text{rank}_{\mathbb{F}_2}(\boldsymbol{Y}_2) = 2$.

By increasing the size of the data matrix, one can also find cases where the $\text{rank}(\boldsymbol{Y}) < \text{rank}_{\{0,1\}}(\boldsymbol{Y})$. For example, for [29]

$$\boldsymbol{Y}_3 = \begin{bmatrix} 1\,1\,0\,0 \\ 1\,0\,1\,0 \\ 0\,1\,0\,1 \\ 0\,0\,1\,1 \end{bmatrix},$$

we have $\text{rank}(\boldsymbol{Y}_3) = 3$, while $\text{rank}_{\{0,1\}}(\boldsymbol{Y}_3) = \text{rank}_{\mathbb{B}}(\boldsymbol{Y}_3) = 4$.

When the rank-one components $\boldsymbol{X}^r$ have disjoint supports, all of the previously mentioned ranks coincide. Thus, for [23]

$$\boldsymbol{Y}_4 = \begin{bmatrix} 1\,1\,0\,0 \\ 1\,1\,0\,0 \\ 0\,0\,1\,1 \\ 0\,0\,1\,1 \end{bmatrix}, \tag{6}$$

$\text{rank}_{\{0,1\}}(\boldsymbol{Y}_4) = \text{rank}_{\mathbb{B}}(\boldsymbol{Y}_4) = \text{rank}_{\mathbb{F}_2}(\boldsymbol{Y}_4) = \text{rank}(\boldsymbol{Y}_4) = 2$, and one exact decomposition corresponding to this rank has factors

$$\boldsymbol{A} = \boldsymbol{B} = \begin{bmatrix} 1\,0 \\ 1\,0 \\ 0\,1 \\ 0\,1 \end{bmatrix}, \tag{7}$$

for all factorizations presented above, but also for factorizations on $\mathbb{R}$.

In data mining applications, for interpretability reasons, it is expected that factors $\boldsymbol{A}$ and $\boldsymbol{B}$ could be retrieved uniquely from the data up to joint column permutations. It is well-known that, in general, matrix factorization over the real numbers is not unique. For F2MF, the factorization is unique only when $R = 1$, since for $R \geqslant 2$ one can find binary $\boldsymbol{T}$, $\boldsymbol{T}'$, different from permutation matrices, such that $\boldsymbol{A} \odot \boldsymbol{B}^{\mathsf{T}} = (\boldsymbol{A} \odot \boldsymbol{T}) \odot (\boldsymbol{B} \odot \boldsymbol{T}')^{\mathsf{T}}$. Regarding bMF and BMF, the binary constraints on $\boldsymbol{A}$ and $\boldsymbol{B}$ allow to retrieve unique factors under some particular conditions (see [23, 28, 30–32] for details on uniqueness conditions).

### *2.2. Polynomial representation of a general Boolean function*

We focus in this paper in solving problem (3) where $f(\cdot)$ is a general Boolean function $f : \{0, 1\}^R \to \{0, 1\}$.

We consider a two-step approach: in the first step, we apply an optimization algorithm to solve a relaxed version of (3) where the binary constraints are dropped. The elements of $\boldsymbol{A}$ and $\boldsymbol{B}$ are

either allowed to lie on $\mathbb{R}$ or on the interval [0, 1]. In the second step, the resulting approximations of $\boldsymbol{A}$ and $\boldsymbol{B}$, denoted $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{B}}$, are projected onto the set of binary values. This projection $\mathcal{P}_{\mathbb{B}}(\cdot)$ corresponds simply to a thresholding operation. For example, for the elements of $\hat{\boldsymbol{A}}$:

$$\left[\mathcal{P}_{\mathbb{B}}(\hat{\boldsymbol{A}})\right]_{ir} = \begin{cases} 0 & , \text{ for } \hat{a}_{ir} < 0.5, \\ 1 & , \text{ for } \hat{a}_{ir} \geqslant 0.5. \end{cases} \tag{8}$$

To apply this approach, the Boolean function $f(\cdot)$ must be represented by another function $\bar{f}$ defined for real inputs. $\bar{f}$ should be defined in such a way that both functions are equal for binary inputs. In this work, we define $\bar{f}(\cdot)$ relying on the fact that any Boolean function of $R$ variables $\mathbf{x} = [x_1, x_2, \cdots, x_R]^{\mathsf{T}}$ can be written as a multivariate polynomial. For any $\mathbf{x} \in \{0, 1\}^R$, the following polynomial achieves the same values as $f(\cdot)$ [33]:

$$\bar{f}(\mathbf{x}) = \sum_{\mathbf{w} \in \mathcal{W}^1} \left\{ \prod_{i|w_i=1} x_i \prod_{j|w_j=0} (1 - x_j) \right\}, \tag{9}$$

where $\mathcal{W}^1 = \left\{ \mathbf{w} \in \{0,1\}^R \,|\, f(\mathbf{w}) = 1 \right\}$.

Examples of polynomial representation of simple Boolean functions are the following:

- Logical 'OR' with $R = 2$, $(x_1 \vee x_2)$:

$$\bar{f}_{\text{OR}}(x_1, x_2) = (1 - x_1)x_2 + x_1(1 - x_2) + x_1 x_2. \tag{10}$$

- Logical 'XOR' with $R = 2$, $(x_1 \oplus x_2)$:

$$\bar{f}_{\text{XOR}}(x_1, x_2) = (1 - x_1)x_2 + x_1(1 - x_2). \tag{11}$$

- 3-term majority:

$$\begin{aligned} \bar{f}_{\text{MAJ}}(x_1, x_2, x_3) = \mathbb{1}_{(\sum_i x_i) \geqslant 2}(x_1, x_2, x_3) = (1 - x_1)x_2 x_3 + x_1(1 - x_2)x_3 \\ + x_1 x_2(1 - x_3) + x_1 x_2 x_3. \end{aligned} \tag{12}$$

Observe that with this representation, the number of terms in the polynomial depends on the cardinal of $\mathcal{W}^1$ (number of input combinations such that $f(\mathbf{x}) = 1$). If the set $\mathcal{W}^0 = \left\{ \mathbf{w} \in \{0,1\}^R \,|\, f(\mathbf{w}) = 0 \right\}$ has a smaller cardinal than $\mathcal{W}^1$, then it may be more convenient to use another equivalent form of $\bar{f}(\cdot)$:

$$\bar{f}(\mathbf{x}) = 1 - \sum_{\mathbf{w} \in \mathcal{W}^0} \left\{ \prod_{i|w_i=1} x_i \prod_{j|w_j=0} (1 - x_j) \right\}. \tag{13}$$

Note that in the case of $x_1 \vee x_2$, the representation above leads to $\bar{f}_{\mathrm{OR}}(x_1, x_2) = 1-(1-x_1)(1-x_2)$ and the corresponding $R$-term version of 'OR' has a simple expression:

$$\bar{f}_{\mathrm{OR}}(x_1, x_2, \cdots, x_R) = 1 - \prod_{r=1}^{R}(1 - x_r). \tag{14}$$

In the rest of the paper, we use representation (13), since it allows an easier presentation of the algorithms that we propose in the specific case of BMF.

## 3. Algorithms

Two properties of $\bar{f}(\cdot)$ are interesting from an optimization point of view: this function is differentiable and it is multilinear in its inputs. Since $\bar{f}(\cdot)$ is differentiable, gradient descent can be applied to attempt solving the corresponding relaxed versions of (3). Multilinearity of $\bar{f}(\cdot)$ with respect to its inputs implies that this function is also multilinear in the columns of $\boldsymbol{A}$ and $\boldsymbol{B}$. Therefore, if we use a block-coordinate descent approach to attempt minimizing relaxed (3), and we set the blocks of variables to be the columns of $\boldsymbol{A}$ and $\boldsymbol{B}$, the block updates will be given by the solutions of simple linear least squares problems. As a consequence, each of these properties leads to a different algorithm for solving relaxed (3). These algorithms are detailed next.

### *3.1. Gradient descent (GD) algorithm*

In the first algorithm, we consider a relaxed version of (3) where elements of the factors are allowed to lie in $\mathbb{R}$. To force the solution to be close to binary, we introduce a penalty term $\mathcal{G}(\boldsymbol{A}, \boldsymbol{B})$ in the objective function as in [16, 23]. The expression of this penalty term is

$$\mathcal{G}(\boldsymbol{A}, \boldsymbol{B}) = \frac{1}{2}\left\{\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{r=1}^{R}\left[a_{ir}^2(1-a_{ir})^2 + b_{jr}^2(1-b_{jr})^2\right]\right\}. \tag{15}$$

Note that this penalty is minimal whenever all elements of the factors are '0' or '1'. The new optimization problem we have to solve is the following:

$$\begin{array}{ll} \text{minimize} & \mathcal{H}(\boldsymbol{A}, \boldsymbol{B}; \lambda) = \mathcal{F}(\boldsymbol{A}, \boldsymbol{B}) + \lambda\mathcal{G}(\boldsymbol{A}, \boldsymbol{B}) \\ \text{with respect to} & \boldsymbol{A} \in \mathbb{R}^{I\times R}, \boldsymbol{B} \in \mathbb{R}^{J\times R}, \end{array} \tag{16}$$

where $\lambda > 0$ is a given value. Since this penalty term is differentiable, $\mathcal{H}(\boldsymbol{A}, \boldsymbol{B}; \lambda)$ is differentiable. Therefore, we can apply the standard gradient descent algorithm to find its critical points.

In standard gradient descent, the entries of the parameters vector $\boldsymbol{\theta} = \left[\mathrm{vec}(\boldsymbol{A})^{\mathsf{T}}\, \mathrm{vec}(\boldsymbol{B})^{\mathsf{T}}\right]^{\mathsf{T}}$, where $\mathrm{vec}(\boldsymbol{A}) = \left[\begin{array}{c} \boldsymbol{a}_1^{\mathsf{T}}, \boldsymbol{a}_2^{\mathsf{T}}, \cdots, \boldsymbol{a}_R^{\mathsf{T}} \end{array}\right]^{\mathsf{T}}$ and $\mathrm{vec}(\boldsymbol{B}) = \left[\begin{array}{c} \boldsymbol{b}_1^{\mathsf{T}}, \boldsymbol{b}_2^{\mathsf{T}}, \cdots, \boldsymbol{b}_R^{\mathsf{T}} \end{array}\right]^{\mathsf{T}}$, are estimated jointly. The

estimate $\hat{\boldsymbol{\theta}}_k$ of the parameter vector at iterate $k$ is given by

$$\hat{\boldsymbol{\theta}}_k = \hat{\boldsymbol{\theta}}_{k-1} - \gamma_k \, \nabla_{\boldsymbol{\theta}} \mathcal{H}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_{k-1}} \,, \tag{17}$$

where $\gamma_k$ is the step-size of the algorithm and $\nabla_{\boldsymbol{\theta}} \mathcal{H}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_{k-1}}$ is the gradient vector of $\mathcal{H}(\cdot)$ with respect to all parameters $\boldsymbol{\theta}$ evaluated at $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}_{k-1}$.

*Gradient expressions.* The full gradient vector can be written as a function of the partial gradients with respect to $\boldsymbol{A}$ and $\boldsymbol{B}$ as follows

$$\nabla^{\mathsf{T}}_{\boldsymbol{\theta}} \mathcal{H}(\boldsymbol{\theta}) = \begin{bmatrix} \nabla^{\mathsf{T}}_{\mathrm{vec}(\boldsymbol{A})} \mathcal{H}(\boldsymbol{A}, \boldsymbol{B}) & \nabla^{\mathsf{T}}_{\mathrm{vec}(\boldsymbol{A})} \mathcal{H}(\boldsymbol{A}, \boldsymbol{B}) \end{bmatrix} \tag{18}$$

and the partial gradients are

$$\nabla^{T}_{\mathrm{vec}(\boldsymbol{A})} \mathcal{H}(\boldsymbol{A}, \boldsymbol{B}) = \begin{bmatrix} \frac{\partial \mathcal{H}(\boldsymbol{A}, \boldsymbol{B})}{\partial a_{11}} \cdots \frac{\partial \mathcal{H}(\boldsymbol{A}, \boldsymbol{B})}{\partial a_{I1}} \cdots \frac{\partial \mathcal{H}(\boldsymbol{A}, \boldsymbol{B})}{\partial a_{1R}} \cdots \frac{\partial \mathcal{H}(\boldsymbol{A}, \boldsymbol{B})}{\partial a_{IR}} \end{bmatrix}, \tag{19}$$

$$\nabla^{T}_{\mathrm{vec}(\boldsymbol{B})} \mathcal{H}(\boldsymbol{A}, \boldsymbol{B}) = \begin{bmatrix} \frac{\partial \mathcal{H}(\boldsymbol{A}, \boldsymbol{B})}{\partial b_{11}} \cdots \frac{\partial \mathcal{H}(\boldsymbol{A}, \boldsymbol{B})}{\partial b_{J1}} \cdots \frac{\partial \mathcal{H}(\boldsymbol{A}, \boldsymbol{B})}{\partial b_{1R}} \cdots \frac{\partial \mathcal{H}(\boldsymbol{A}, \boldsymbol{B})}{\partial b_{JR}} \end{bmatrix}. \tag{20}$$

The elements of $\nabla_{\mathrm{vec}(\boldsymbol{A})} \mathcal{H}(\boldsymbol{A}, \boldsymbol{B})$ are given by

$$\frac{\partial \mathcal{H}(\boldsymbol{A}, \boldsymbol{B})}{\partial a_{i'r'}} = -\left\{ \sum_{j=1}^{J} \left[ y_{i'j} - \bar{f}(x^{1:R}_{i'j}) \right] \frac{\partial \bar{f}(x^{1:R}_{i'j})}{\partial x^{r'}_{i'j}} b_{jr'} \right\} + \lambda a_{i'r'}(1 - a_{i'r'})(1 - 2a_{i'r'}) \tag{21}$$

for $i' \in \{1, \cdots, I\}$ and $r' \in \{1, \cdots, R\}$, where the expression of the partial derivatives of $\bar{f}(\cdot)$ are

$$\begin{aligned} \frac{\partial \bar{f}(x^{1:R}_{i'j})}{\partial x^{r'}_{i'j}} = & \sum_{\substack{\boldsymbol{w} \in \mathcal{W}^0, \\ w_{r'} = 0}} \left[ \prod_{s|w_s=1} x^{s}_{i'j} \right] \left[ \prod_{\substack{s'|w'_s = 0, \\ s' \neq r'}} (1 - x^{s'}_{i'j}) \right] \\ & - \sum_{\substack{\boldsymbol{w} \in \mathcal{W}^0, \\ w_{r'} = 1}} \left[ \prod_{\substack{s|w_s = 1, \\ s \neq r'}} x^{s}_{i'j} \right] \left[ \prod_{s'|w'_s=0} (1 - x^{s'}_{i'j}) \right]. \end{aligned} \tag{22}$$

The $(j', r')$ element of $\nabla_{\mathrm{vec}(\boldsymbol{B})} \mathcal{H}(\boldsymbol{A}, \boldsymbol{B})$ for $j' \in \{1, \cdots, J\}$, $r' \in \{1, \cdots, R\}$ is

$$\frac{\partial \mathcal{H}(\boldsymbol{A}, \boldsymbol{B})}{\partial b_{j'r'}} = -\left\{ \sum_{i=1}^{I} \left[ y_{ij'} - \bar{f}(x^{1:R}_{ij'}) \right] \frac{\partial \bar{f}(x^{1:R}_{i,j'})}{\partial x^{r'}_{i,j'}} a_{i,r'} \right\} + \lambda b_{j'r'}(1 - b_{j'r'})(1 - 2b_{j'r'}). \tag{23}$$

The partial gradients can be written in vector form as a function of $\boldsymbol{A}$ and $\boldsymbol{B}$ as follows

$$\begin{aligned} \nabla_{\mathrm{vec}(\boldsymbol{A})} \mathcal{H}(\boldsymbol{A}, \boldsymbol{B}) = & -\mathrm{Diag}\left( \boldsymbol{E} \boxdot \mathbf{P}_1, \cdots, \boldsymbol{E} \boxdot \mathbf{P}_R \right) \mathrm{vec}(\boldsymbol{B}) \\ & + \lambda \mathrm{vec}(\boldsymbol{A}) \boxdot (\mathbf{1}_{IR \times 1} - \mathrm{vec}(\boldsymbol{A})) \boxdot (\mathbf{1}_{IR \times 1} - 2\mathrm{vec}(\boldsymbol{A})) \,, \\ \nabla_{\mathrm{vec}(\boldsymbol{B})} \mathcal{H}(\boldsymbol{A}, \boldsymbol{B}) = & -\mathrm{Diag}\left( \boldsymbol{E}^{\mathsf{T}} \boxdot \mathbf{P}^{\mathsf{T}}_1, \cdots, \boldsymbol{E}^{\mathsf{T}} \boxdot \mathbf{P}^{\mathsf{T}}_R \right) \mathrm{vec}(\boldsymbol{A}) \\ & + \lambda \mathrm{vec}(\boldsymbol{B}) \boxdot (\mathbf{1}_{JR \times 1} - \mathrm{vec}(\boldsymbol{B})) \boxdot (\mathbf{1}_{JR \times 1} - 2\mathrm{vec}(\boldsymbol{B})) \,, \end{aligned} \tag{24}$$

where $\boldsymbol{E}$ is the model error matrix

$$\boldsymbol{E} = \boldsymbol{Y} - \bar{f}\left(\boldsymbol{X}^{1:R}\right) = \boldsymbol{Y} - \mathbf{1}_{I\times J} + \sum_{\boldsymbol{w}\in\mathcal{W}^0} \left[\boxdot_{s|w_s=1} \boldsymbol{X}^s\right] \boxdot \left[\boxdot_{s'|w'_s=0} (\mathbf{1}_{I\times J} - \boldsymbol{X}^{s'})\right] \tag{25}$$

and $\boldsymbol{P}_{r'}$ are $I \times J$ matrices given by

$$\begin{aligned}\boldsymbol{P}_{r'} = & \sum_{\substack{\boldsymbol{w}\in\mathcal{W}^0,\\ w_{r'}=0}} \left[\boxdot_{s|w_s=1} \boldsymbol{X}^s\right] \boxdot \left[\boxdot_{\substack{s'|w'_s=0,\\ s'\neq r'}} (\mathbf{1}_{I\times J} - \boldsymbol{X}^{s'})\right] \\ & - \sum_{\substack{\boldsymbol{w}\in\mathcal{W}^0,\\ w_{r'}=1}} \left[\boxdot_{\substack{s|w_s=1,\\ s\neq r'}} \boldsymbol{X}^s\right] \boxdot \left[\boxdot_{s'|w'_s=0} (\mathbf{1}_{I\times J} - \boldsymbol{X}^{s'})\right].\end{aligned} \tag{26}$$

*Step-size, penalty constant and initializations.* In the simplest version of the algorithm the step-size $\gamma_k$ can be set to a small constant value. The penalty factor $\lambda$ may be chosen as variable through iterations: $\lambda$ is set to a value close to zero in the first iterations and its increased up to a high target value.

Since the cost function being minimized is highly nonconvex, gradient descent may converge to spurious critical points. For this reason, it is important to test different initializations of the algorithm and pick the solution which gives the best data fitting. The algorithm can be initialized each time with different random elements for the factor updates $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{B}}$. The elements of $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{B}}$ can be drawn from independent and identically distributed (iid) uniform samples: $\hat{a}_{ir} \sim \mathcal{U}[0,1]$ , $\hat{b}_{ir} \sim \mathcal{U}[0,1]$, for $i \in \{1, \cdots, I\}$, $j \in \{1, \cdots, J\}$ and $r \in \{1, \cdots, R\}$.

### *3.2. Projected Hierarchical Alternating Least Squares (PHALS) algorithm*

In the second approach, named projected hierarchical alternating least squares (PHALS), the cost function is minimized with respect to each $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_R, \boldsymbol{b}_1, \cdots, \boldsymbol{b}_R$ in an alternating manner, similar to the hierarchical alternating least squares (HALS) method [27]. The minimization with respect to each column is performed by relaxing the binary constraints to $\mathbb{R}^I$, $\mathbb{R}^J$. After updating all the estimates of a column of a factor, we project elements of the updated factor onto the interval [0, 1] to prevent the updates to converge to negative or large positive values.

Suppose that we want to update the estimate $\hat{\boldsymbol{a}}_{r'}$ of $\boldsymbol{a}_{r'}$, all other columns of the factors are then considered to be equal to $\hat{\boldsymbol{a}}_r$ with $r \neq r'$ and $\hat{\boldsymbol{b}}_r$ for $r \in \{1, \cdots, R\}$. The updated $\hat{\boldsymbol{a}}_{r'}$ is then given

by the minimization of

$$\mathcal{F}_{\text{PHALS}}(\boldsymbol{a}_{r'}) = \frac{1}{2}\sum_{i=1}^{I}\sum_{j=1}^{J}\left\{y_{ij} - 1 + a_{ir'}\hat{b}_{jr'}\hat{p}_{ij}^{r'} + (1 - a_{ir'}\hat{b}_{jr'})\hat{q}_{ij}^{r'}\right\}^2$$
$$= \frac{1}{2}\sum_{i=1}^{I}\sum_{j=1}^{J}\left[y_{ij} - 1 + \hat{q}_{ij}^{r'} + a_{ir'}\hat{b}_{jr'}(\hat{p}_{ij}^{r'} - \hat{q}_{ij}^{r'})\right]^2, \tag{27}$$

where

$$\hat{p}_{ij}^{r'} = \sum_{\substack{\boldsymbol{w}\in\mathcal{W}^0,\\ w_{r'}=1}}\left[\prod_{\substack{s|w_s=1\\ s\neq r'}}\hat{a}_{is}\hat{b}_{js}\right]\left[\prod_{\substack{s'|w'_s=0\\ s\neq r'}}(1-\hat{a}_{is}\hat{b}_{js})\right] \tag{28}$$

and $\hat{q}_{ij}^{r'}$ is similarly defined except that the summation is done through $\boldsymbol{w}\in\mathcal{W}^0$ whose $w_{r'}=0$.

The cost function $\mathcal{F}_{\text{PHALS}}(\boldsymbol{a}_{r'})$ can be rewritten as $\mathcal{F}_{\text{PHALS}}(\boldsymbol{a}_{r'}) = \sum_{i=1}^{I}\mathcal{F}_i(a_{ir'})$, where, for a given $i'$,

$$\mathcal{F}_{i'}(a_{i'r'}) = \sum_{j=1}^{J}\left(y_{i'j} - 1 + \hat{q}_{i'j}^{r'} + a_{i'r'}\hat{b}_{jr'}(\hat{p}_{i'j}^{r'} - \hat{q}_{i'j}^{r'})\right)^2. \tag{29}$$

Observe that each term $\mathcal{F}_i(a_{ir'})$ of the cost function depends only on one of the $a_{ir'}$, thus the elements of $\hat{\boldsymbol{A}}_{r'}$ can be obtained separately by minimizing $\mathcal{F}_{i'}(a_{i'r'})$. The function $\mathcal{F}_{i'}(a_{i'r'})$ is quadratic on $a_{i'r'}$, therefore its unconstrained minimum can be easily obtained. For a given $i'$, it is

$$\hat{a}_{i'r'} = \frac{\sum_{j=1}^{J}\left[(y_{i'j} - 1 + \hat{q}_{i'j}^{r'})(\hat{q}_{i'j}^{r'} - \hat{p}_{i'j}^{r'})\hat{b}_{jr'}\right]}{\sum_{j=1}^{J}\left[\hat{b}_{jr'}(\hat{q}_{i'j}^{r'} - \hat{p}_{i'j}^{r'})\right]^2}. \tag{30}$$

Once $\hat{\boldsymbol{a}}_{r'}$ has been completely updated, the arrays $\hat{p}_{i,j}^{r}$ and $\hat{q}_{ij}^{r}$ have to be recalculated for the update of the next $\hat{\boldsymbol{a}}_r$. When all columns of $\hat{\boldsymbol{A}}$ have been updated, the projection of its elements onto $[0,\,1]$ is given by

$$\left[\mathcal{P}_U(\hat{\boldsymbol{A}})\right]_{ir} = \begin{cases} 0 & , \text{ for } \hat{a}_{ir} < 0, \\ \hat{a}_{ir} & , \text{ for } 0 \leqslant \hat{a}_{ir} \leqslant 1, \\ 1 & , \text{ for } \hat{a}_{ir} > 1. \end{cases} \tag{31}$$

A similar procedure is applied to the updates $\hat{\boldsymbol{b}}_r$. The updates before projection of its elements for $j' \in \{1,\,2,\,\cdots,\,J\}$ are

$$\hat{b}_{j'r'} = \frac{\sum_{i=1}^{I}\left[(y_{ij'} - 1 + \hat{q}_{ij'}^{r'})(\hat{q}_{ij'}^{r'} - \hat{p}_{ij'}^{r'})\hat{a}_{ir'}\right]}{\sum_{i=1}^{I}\left[\hat{a}_{ir'}(\hat{q}_{ij'}^{r'} - \hat{p}_{ij'}^{r'})\right]^2}. \tag{32}$$

After executing $K$ updates of all columns of $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{B}}$ with PHALS, the elements of $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{B}}$ are projected on the set of binary values with $\mathcal{P}_{\mathbb{B}}(\cdot)$ (8).

The full algorithm using PHALS to obtain an approximate generalized Boolean factorization is given in Algorithm 1.

### *3.3. Algorithms for BMF*

In the specific case of BMF, the most popular Boolean factorization used in practice, the expressions of different quantities of the underlying algorithms can be easily written for any $R$. In this subsection, we detail these expressions.

By relying on (14), BMF can be written in matrix form as follows:

$$\bar{f}\left(\boldsymbol{X}^{1:R}\right) = \bigvee_{r=1}^{R} \boldsymbol{X}^r = \mathbf{1}_{I\times J} - \boxed{\cdot}_{r=1}^{R} \left(\mathbf{1}_{I\times J} - \boldsymbol{X}^r\right) \tag{33}$$

For given $\boldsymbol{X}^{1:R}$, the reconstruction error $\boldsymbol{E}$ can be written as

$$\boldsymbol{E} = \boldsymbol{Y} - \mathbf{1}_{I\times J} + \boxed{\cdot}_{r=1}^{R} \left(\mathbf{1}_{I\times J} - \boldsymbol{X}^r\right) = \boldsymbol{Y} - \mathbf{1}_{I\times J} + \boxed{\cdot}_{r=1}^{R} \left(\mathbf{1}_{I\times J} - \boldsymbol{a}_r \boldsymbol{b}_r^{\mathsf{T}}\right). \tag{34}$$

The $\boldsymbol{P}_{r'}$ matrices (26) required in GD are

$$\boldsymbol{P}_{r'} = \boxed{\cdot}_{\substack{s=1 \\ s\neq r'}}^{R} \left(\mathbf{1}_{I\times J} - \boldsymbol{X}^s\right) = \boxed{\cdot}_{\substack{s=1 \\ s\neq r'}}^{R} \left(\mathbf{1}_{I\times J} - \boldsymbol{a}_s \boldsymbol{b}_s^{\mathsf{T}}\right). \tag{35}$$

For PHALS, the quantities that vary depending on the choice of the Boolean function $\bar{f}(\cdot)$ are $\hat{p}_{ij}^{r'}$ and $\hat{q}_{ij}^{r'}$. For BMF, we have $\hat{p}_{ij}^{r'} = 0$ for all possible tuples $(ijr')$, while $\hat{q}_{ij}^{r'}$ can be written in matrix form as $\boldsymbol{P}_{r'}$ above for $r' \in \{1, \cdots, R\}$:

$$\boldsymbol{Q}_{r'} = \boldsymbol{P}_{r'} = \boxed{\cdot}_{\substack{s=1 \\ s\neq r'}}^{R} \left(\mathbf{1}_{I\times J} - \boldsymbol{a}_s \boldsymbol{b}_s^{\mathsf{T}}\right). \tag{36}$$

## 4. Numerical experiments

In this section, we present the results of numerical experiments concerning the proposed algorithms. We focus first on the BMF setting, that is, when the combining function $f(\cdot)$ is the 'OR' function. Under this setting, we compare the performance of the algorithms with 3 other BMF methods from the literature on simulated noisy binary data. In the first and second simulation

**Algorithm 1** Projected hierarchical alternating least squares for general Boolean factorization (PHALS)

**Require:** $\boldsymbol{Y}$, $R$, $K$.

Initialize $\hat{\boldsymbol{A}}$, $\hat{\boldsymbol{B}}$ with random i.i.d. uniform elements $\hat{a}_{ir} \sim \mathcal{U}[0,1]$ , $\hat{b}_{ir} \sim \mathcal{U}[0,1]$, for $i \in \{1, \cdots, I\}$, $j \in \{1, \cdots, J\}$ and $r \in \{1, \cdots, R\}$.
**for** $k \in \{1, \cdots, K\}$ **do**
  Update $\hat{\boldsymbol{A}}$:
  **for** $r \in \{1, \cdots, R\}$ **do**
    Update each column of $\hat{\boldsymbol{A}}$:
    **for** $i \in \{1, \cdots, I\}$ **do**
      Update elements of $\hat{\boldsymbol{a}}_r$ (30): $\hat{a}_{i'r'} = \dfrac{\sum_{j=1}^{J} \left[ (y_{i'j} - 1 + \hat{q}_{i'j}^{r'})(\hat{q}_{i'j}^{r'} - \hat{p}_{i'j}^{r'}) \hat{b}_{jr'} \right]}{\sum_{j=1}^{J} \left[ \hat{b}_{jr'} (\hat{q}_{i'j}^{r'} - \hat{p}_{i'j}^{r'}) \right]^2}$
    **end for**
    **for** $r' \in \{1, \cdots R\}$ and $r' \neq r$ **do**
      Update $\hat{p}_{ijr'}$ with (28) for $r' \neq r$ and $\hat{q}_{ijr'}$ in a similar manner.
    **end for**
  **end for**
  Project onto [0, 1] (31): $\hat{\boldsymbol{A}} := \mathcal{P}_U(\hat{\boldsymbol{A}})$
  Update $\hat{\boldsymbol{B}}$:
  **for** $r \in \{1, \cdots, R\}$ **do**
    Update each column of $\hat{\boldsymbol{B}}$:
    **for** $i \in \{1, \cdots, J\}$ **do**
      Update elements of $\hat{\boldsymbol{b}}_r$ (32): $\hat{b}_{j'r'} = \dfrac{\sum_{i=1}^{I} \left[ (y_{ij'} - 1 + \hat{q}_{ij'}^{r'})(\hat{q}_{ij'}^{r'} - \hat{p}_{ij'}^{r'}) \hat{a}_{ir'} \right]}{\sum_{i=1}^{I} \left[ \hat{a}_{ir'} (\hat{q}_{ij'}^{r'} - \hat{p}_{ij'}^{r'}) \right]^2}$
    **end for**
    **for** $r' \in \{1, \cdots R\}$ and $r' \neq r$ **do**
      Update $\hat{p}_{ijr'}$ and $\hat{q}_{ijr'}$.
    **end for**
  **end for**
  Project onto [0, 1] (31): $\hat{\boldsymbol{B}} := \mathcal{P}_U(\hat{\boldsymbol{B}})$
**end for**
Project onto {0, 1} (8): $\hat{\boldsymbol{A}} := \mathcal{P}_{\mathbb{B}}(\hat{\boldsymbol{A}}), \quad \hat{\boldsymbol{B}} := \mathcal{P}_{\mathbb{B}}(\hat{\boldsymbol{B}})$

settings, the data are drawn from random BMF models which are then perturbed by binary flipping noise. In the first setting, the performance of the algorithms is presented for different number of columns $R$ of $\boldsymbol{A}$ and $\boldsymbol{B}$, and the probability of binary flipping the data (equivalent to noise intensity) is kept constant. In the second setting, $R$ is kept constant and results are shown for different values of the probability of binary flipping. Simulation results will then be presented concerning the sensitivity of the proposed methods to initialization, convergence behavior and time complexity. The presentation of simulation results on BMF is then followed by its application to real datasets. We apply PHALS to retrieve the BMF of the following datasets: the congressional voting dataset [34][1], the zoo dataset [34][2], the New and Old Worlds (NOW) paleontological database [35] and the United Nations voting dataset [36][3]. We end the section by presenting performance results for PHALS and GD for simulated data generated with $f(\cdot)$ equal to the XOR with two inputs (XOR-2) and to the 3-term majority function (MAJ-3).

### *4.1. Performance for different R*

In what follows, the performances of the two proposed algorithms PHALS and GD are compared to 3 state-of-the-art methods for BMF discussed in the introduction: *ASSOciation rules* algorithm (ASSO) [18], the *Formal Concept* (FC) analysis based algorithm [19] and the *Post-NonLinear Penalty Function* (PNL-PF) algorithm [23]. We have also included in the simulations a version of GD that is initialized with PHALS, we will denote that version of GD as PHALS+GD.

In this set of simulations, the data matrix $\boldsymbol{Y}$ is a $20 \times 20$ matrix corresponding to a perturbed version of a BMF with $R$ components $\boldsymbol{X} = \bigvee_{r=1}^{R} \boldsymbol{a}_r \boldsymbol{b}_r^{\mathsf{T}}$. The noise matrix is binary $\boldsymbol{N} \in \{0, 1\}$ and the perturbation consists in flipping the elements of $\boldsymbol{X}$. Therefore, the elements of $\boldsymbol{Y}$ can be written using the logical 'XOR': $y_{ij} = x_{ij} \oplus n_{ij}$. The elements of $\boldsymbol{N}$ are drawn iid from a Bernoulli distribution $n_{ij} \sim \mathcal{B}(p_n)$ where $p_n = \mathbb{P}(n_{ij} = 1)$.

For a given data matrix $\boldsymbol{Y}$, PHALS and GD are initialized at random $n_{\text{init}} = 3$ times and the solution achieving the least reconstruction error $\mathcal{F}(\hat{\boldsymbol{A}}, \hat{\boldsymbol{B}})$ is kept. For PHALS+GD, GD is initialized with the best of the 3 initializations of PHALS. PNL-PF is initialized with the result of NMF [22] applied to the data. The NMF algorithm is initialized randomly as PHALS and GD. ASSO and FC do not require initializations.

[1] `https://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records`

[2] `https://archive.ics.uci.edu/ml/datasets/Zoo`

[3] `https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/LEJUQZ`

GD and PNL-PF are executed with $K_{\text{GD/PNL-PF}} = 2000$ iterations for each simulation, while PHALS is executed $K_{\text{PHALS}} = 1000$ iterations. The step-length of GD is set to a constant $\gamma_k = \gamma = 0.1$ and its hyperparameter $\lambda$ is increased linearly from 0.01 to 10. The threshold value $\tau$ for ASSO (see [18]) is fixed to 0.9.

Under each different simulation setting, $N = 100$ random data matrices $\boldsymbol{Y}^n$ $(n = 1, \ldots, N)$ are generated, each with a random pair $(\boldsymbol{X}^n, \boldsymbol{N}^n)$. Each $\boldsymbol{X}^n$ with a given $R$ is obtained from randomly generated factor matrices $\boldsymbol{A}^n$ and $\boldsymbol{B}^n$. The elements of $\boldsymbol{A}^n$ and $\boldsymbol{B}^n$ are iid samples from Bernoulli distributions $a_{ir} \sim \mathcal{B}(p_a)$, $b_{ir} \sim \mathcal{B}(p_b)$ where $p_a = p_b = 0.4$. After applying the algorithms on all $\boldsymbol{Y}^n$, their performances in terms of normalized mean square errors (NMSE) of prediction of $\boldsymbol{X}^n$ and retrieval of $\boldsymbol{A}^n$ and $\boldsymbol{B}^n$ are evaluated. The expressions of these NMSE are the following: $\text{NMSE}_{\boldsymbol{X}} = \frac{1}{NIJ} \sum_{n=1}^{N} \left\| \boldsymbol{X}^n - \bigvee_{r=1}^{R} \hat{\boldsymbol{a}}_r^n \left(\hat{\boldsymbol{b}}_r^n\right)^{\mathsf{T}} \right\|_F^2$, $\text{NMSE}_{\boldsymbol{A}} = \frac{1}{NIR} \sum_{n=1}^{N} \left\| \boldsymbol{A}^n - \hat{\boldsymbol{A}}^n \right\|_F^2$, $\text{NMSE}_{\boldsymbol{B}} = \frac{1}{NJR} \sum_{n=1}^{N} \left\| \boldsymbol{B}^n - \hat{\boldsymbol{B}}^n \right\|_F^2$. Since the elements of all matrices are binary, these NMSE can be interpreted as error rates. Note that, due to the permutation ambiguity on the estimation of the factors, their columns should be permuted to match in the best possible way those of the true factors before the evaluation of the NMSE.

The evolution of the NMSE for the BMF methods is shown in Fig. 1a for $p_n = 0.1$ and $R \in \{2, \cdots, 6\}$. As intuitively expected, for most methods, larger values of $R$ lead to larger NMSE both on prediction of $\boldsymbol{X}$ and on the retrieved factors. One can also observe that there is no significant difference in performance between the PHALS, GD and PHALS+GD. PNL-PF has a moderately inferior performance compared to the proposed methods, while ASSO and FC achieve a significantly inferior performance in terms of retrieving $\boldsymbol{X}$, $\boldsymbol{A}$ and $\boldsymbol{B}$. ASSO and FC do not seem to be adapted to the approximation setting where noise is present, which has been also observed in previous studies [23].

### *4.2. Results for different $p_n$*

The second simulation setting is very similar to the previously presented one, except that, in this case, $R$ is kept to a constant value, $R = 3$, and $p_n$ is varied from 0 to 0.3 by increments of 0.02. Fig. 1b displays the performances of the methods. One can see that, as it is naturally expected, the performances degrade as $p_n$ increases. As in the previous setting, all the proposed methods seem to lead to similar performances and they achieve a superior performance compared to the other 3 methods from the literature. Observe also that as $p_n$ gets close to 0.3 all NMSE of the proposed methods are close to 0.3 and for values smaller than $p_n < 0.3$ the NMSE seem to be smaller than $p_n$.

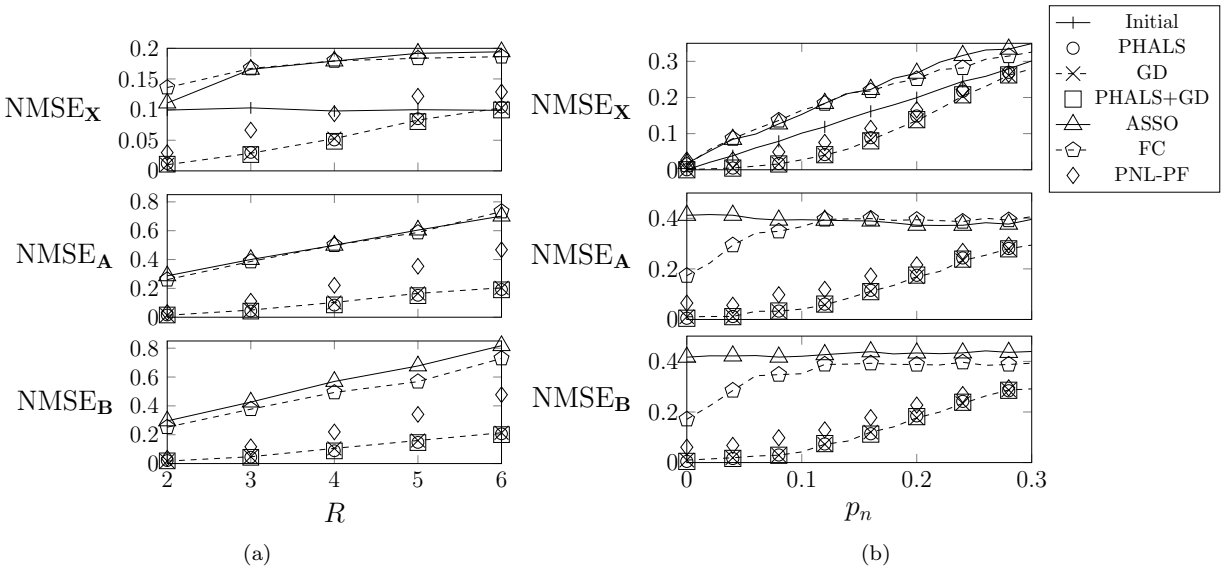



Figure 1: NMSE for the prediction of $\boldsymbol{X}$ of size $20 \times 20$ and retrieval of its BMF factors $\boldsymbol{A}$ and $\boldsymbol{B}$. The factorization algorithms are executed on a noisy version $\boldsymbol{Y}$ of $\boldsymbol{X}$. The noise acts by flipping the elements of $\boldsymbol{Y}$ with probability $p_n$. The curves named "Initial" in the top subfigures indicate the NMSE of predicting $\boldsymbol{X}$ simply using $\boldsymbol{Y}$. In (a), the number of columns of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ varies from 2 to 6, while $p_n = 0.1$. In (b), $R = 3$ and $p_n$ is varied from 0 to 0.3 with increments of 0.02.

Note, on the top figure, that predicting $\boldsymbol{X}$ using the noisy data is as efficient as using the proposed methods for $p_n = 0.3$.

Finally, one can see that, when $p_n = 0$, the NMSE on the factors is not zero. This may be due to convergence of the algorithms to factorizations which are not global minima of (16) or to the non uniqueness of the approximation of some realizations $\boldsymbol{Y}^n$. However, since NMSE are very small for $p_n = 0$, it seems that the occurrence of such issues is very rare.

### *4.3. Simple example with unique decomposition*

From the previous simulation results, it seems that ASSO and FC are not adequate in the BMF approximation setting. Therefore, in what follows, we focus on comparing only PHALS, GD and PNL-PF.

As previously presented, the algorithms may converge to wrong $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{B}}$, even in the exact factorization setting ($p_n = 0$) when the underlying factorization is unique. Due to the non convexity of the underlying cost functions, not all initializations $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{B}}$ lead to the original factors. To try to assess to which extent the algorithms are prone to this behavior, we have tested them on 3 small

| | $\boldsymbol{Y}_1$ | $\boldsymbol{Y}_4$ | $\boldsymbol{Y}_5$ |
|---|---|---|---|
| PHALS | 99 | 100 | 96 |
| GD | 79 | 100 | 48 |
| PHALS+GD | 99 | 100 | 96 |
| PNL-PF | 100 | 66 | 72 |

Table 2: Success rate $S_{\%}$ for retrieving the exact BMF for 3 different matrices $\boldsymbol{Y}_1$, $\boldsymbol{Y}_4$ and $\boldsymbol{Y}_5$ using $n_{\text{init}} = 100$ different random initializations.

exact factorization problems with unique factorizations. The 3 considered data matrices are $\boldsymbol{Y}_1$ (5), $\boldsymbol{Y}_4$ (6) and the following $5 \times 5$ matrix from [23]:

$$\boldsymbol{Y}_5 = \begin{bmatrix} 1\,1\,0\,0\,0 \\ 1\,1\,0\,0\,0 \\ 0\,1\,1\,1\,0 \\ 0\,1\,1\,1\,1 \\ 0\,1\,1\,0\,1 \end{bmatrix}.$$

The matrices $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_4$ have rank 2, while $\boldsymbol{Y}_3$ has rank 3. PHALS, GD and PNL-PF are applied to these data with $n_{\text{init}} = 100$ random initializations $\hat{\boldsymbol{A}}_0^i$, $\hat{\boldsymbol{B}}_0^i$, $i \in \{1, \cdots, n_{\text{init}}\}$. The number of iterations for all algorithms is $K = 2000$. Parameters $\gamma_k$ and $\lambda$ of GD have been set as in the previous simulations. For each algorithm and each data matrix we have calculated the success rate $S_{\%}$ in percent of retrieving $\boldsymbol{A}$ and $\boldsymbol{B}$ from data: $S_{\%} = \left[\text{card}\left(\left\{i | \hat{\boldsymbol{A}}^i = \boldsymbol{A} \text{ and } \hat{\boldsymbol{B}}^i = \boldsymbol{B}\right\}\right)/n_{\text{init}}\right] \times 100$, where $\text{card}(\cdot)$ denotes the cardinal of a set and $\hat{\boldsymbol{A}}^i$, $\hat{\boldsymbol{B}}^i$ are the output factors of an algorithm when initialized with $\hat{\boldsymbol{A}}_0^i$ and $\hat{\boldsymbol{B}}_0^i$. The success rates are displayed in Tab. 2. We observe that the algorithm which seem less prone to converge to spurious factors is PHALS. GD and PNL-PF may converge to spurious factors, but they do not seem to behave equally through the examples. From the results, we can also note that applying GD initialized with the resulting factors from PHALS does not lead to an improvement in the success rate.

### *4.4. Convergence*

To compare the convergence behavior of PHALS, GD and PNL-PF, we generate 3 random $\boldsymbol{Y}^k$ in the same manner as in Subsec. 4.1 for $R$ equal to 2, 4 and 6. We then apply the 3 algorithms with $n_{\text{init}} = 100$ and $K = 500$. At each iteration $k \in \{1, \cdots, K\}$, we evaluate the overall changes in the factors using the following quantity: $\Delta_k = \frac{\|\hat{\boldsymbol{A}}_k - \hat{\boldsymbol{A}}_{k-1}\|_F^2 + \|\hat{\boldsymbol{B}}_k - \hat{\boldsymbol{B}}_{k-1}\|_F^2}{\|\hat{\boldsymbol{A}}_0\|_F^2 + \|\hat{\boldsymbol{B}}_0\|_F^2}$, where $\hat{\boldsymbol{A}}_k$ and $\hat{\boldsymbol{B}}_k$ are the $k$-th updates of the factors for a given algorithm. The quantity $\Delta_k$ is small whenever the factors do not change in consecutive iterations. Therefore, if $\Delta_k$ reduces as $k$ increases, the algorithm is converging. Fig. 2 shows some statistics on $\Delta_k$ for each algorithm. The statistics displayed are

the median, the 5-th and 95-th percentiles of $\Delta_k$ evaluated with the $n_{\text{init}} = 100$ available values for each $k$. The overall behavior we can observe from this figure is that PHALS is the fastest method in terms of convergence, while GD is the slowest. One can also note that the 95-th percentile of $\Delta_k$ increases as $R$ increases. Although PHALS is much faster than the other methods to converge, when $R$ increases, some initializations may lead it to converge slowly or not to converge at all.

*Remarks on convergence guarantees:* in their present form, we are not able to give theoretical guarantees on convergence of the iterates of GD and PHALS.

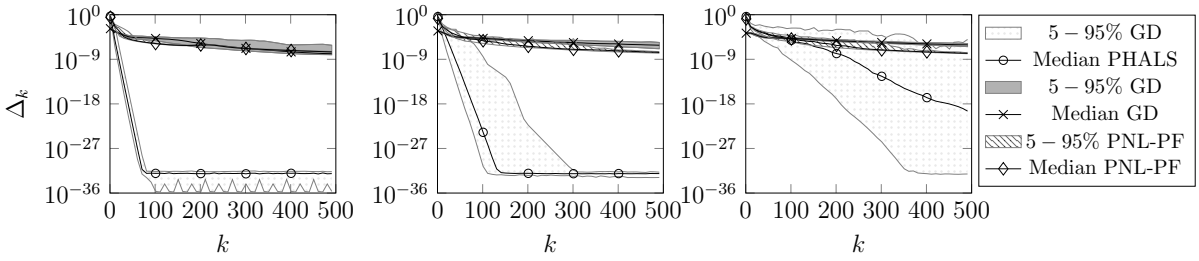
Concerning GD, as presented above, it seems that in practice the algorithm converges, if the constant step-size $\gamma_k = \gamma$ is chosen sufficiently small. However, theoretical guarantees for convergence of GD require that a global Lipschitz constant of the gradient of the objective function exists. Unfortunately, this does not seem to be true for the objective in (16). A possible way to ensure convergence is to use an adaptive step-length $\gamma_k$ given by backtracking line-search [37]. With this modification, since the cost function is analytic and coercive, convergence of GD is guaranteed using the results from [38].

PHALS is a block coordinate descent algorithm. For this class of algorithms, convergence of the iterates can be ensured, for example, using the results of [39]. To use the results of [39], the objective function should be separately strongly convex in each block of variables $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_R, \boldsymbol{b}_1, \cdots, \boldsymbol{b}_R$. Unfortunately, this cannot be guaranteed, and, in practice, one can see that for difficult factorization cases (large $R$), some initializations may lead to non-converging iterates. One possibility to solve this issue is to add proximal terms to the objective function at each update. For the updates of $\boldsymbol{a}_r$, one should add $\rho\|\boldsymbol{a}_r - \hat{\boldsymbol{a}}_r\|_2^2$, where $\rho > 0$ is a pre-defined constant and $\hat{\boldsymbol{a}}_r$ is the most recent update of $\boldsymbol{a}_r$. Similarly, for the updates $\boldsymbol{b}_r$, the term $\rho\|\boldsymbol{b}_r - \hat{\boldsymbol{b}}_r\|_2^2$ should be added. With modified updates considering this additional terms, it may be possible to use the results of [39] to ensure convergence of the iterates.

### *4.5. Time complexity*

Using a similar simulation setting from the previous subsection, we have also measured the execution time for the 3 algorithms to finish 2000 iterations. The statistics on execution time[4] for 100 runs of the algorithms and for $R \in \{2, \cdots, 6\}$ are shown in Tab. 3. We observe that PNL-PF takes much less time than the other methods. GD is from 10 to 40 times slower than PNL-PF and we

[4]These simulations are realized in *Scilab* version 6.1.0 with a processor Intel®Core™ i7-7820HQ, 2.90GHz and with 16GB of RAM.




(a) $R = 2$ (b) $R = 4$ (c) $R = 6$

Figure 2: Statistics about the overall changes $\Delta_k$ (4.4) in the updates $\hat{\boldsymbol{A}}_k$ and $\hat{\boldsymbol{B}}_k$ for GD, PHALS and PNL-PF for the approximate factorization of 3 noisy binary matrices of size $20 \times 20$. Each matrix is generated by randomly drawing a binary BMF model with a given $R$, then applying binary flipping noise. The subfigures are generated with different values of $R$, they are indicated in the subcaptions. The statistics are evaluated for $n_{\text{init}} = 100$ different random initializations. The extremes of the bands $(5\% - 95\%)$ around the median are the 5-th and 95-th percentiles of $\Delta_k$.

| | | $R = 2$ | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| PHALS | 5-th percentile | 0.844 | 1.547 | 2.188 | 3.0313 | 4.281 |
| | Median | 0.703 | 1.297 | 2.000 | 2.906 | 3.938 |
| | 95-th percentile | 0.610 | 1.219 | 1.813 | 2.781 | 3.750 |
| GD | 5-th percentile | 1.734 | 2.734 | 3.438 | 24.438 | 32.688 |
| | Median | 1.547 | 2.359 | 3.281 | 23.422 | 30.500 |
| | 95-th percentile | 1.406 | 2.219 | 3.141 | 22.313 | 29.125 |
| PNL-PF | 5-th percentile | 0.203 | 0.250 | 0.234 | 1.000 | 0.984 |
| | Median | 0.156 | 0.172 | 0.172 | 0.672 | 0.719 |
| | 95-th percentile | 0.094 | 0.125 | 0.125 | 0.422 | 0.469 |

Table 3: Statistics on total execution times in seconds for approximate $R$-component BMF of $20 \times 20$ binary matrices. The statistics are evaluated for 100 runs of the 3 algorithms with $K = 2000$ iterations in each run.

can clearly see a large relative increase in execution for GD when passing from $R = 4$ to $R = 5$. Such an increase can also be observed in a lesser extent in PNL-PF, while in PHALS the relative increase is smaller than a factor of 2. Since real datasets may be of sizes much larger than $20 \times 20$, it is clear from these simulations that GD cannot be reasonably used in practical data analysis problems with the implementation used in this work.

### *4.6. Discussion on the results*

In terms of approximation performance PHALS leads to better results than PNL-PF at the expense of a longer execution time per iteration and of a risk of producing slowly or non converging

iterations for large values of $R$ (in our simulations mainly for $R \geqslant 6$). Note however that for small $R$, $\Delta_k$ for PHALS decreases much faster than for PNL-PF, thus if a threshold on $\Delta_k$ is used as convergence criterion to stop the algorithm, the longer execution times of PHALS iterations are compensated by its much faster convergence.

### *4.7. Real datasets*

In what follows, we obtain the approximate BMF of different real binary datasets. From the previous results on factorization performance obtained through simulations, factorization results are expected to be mostly similar for PHALS, GD and PNL-PF. Therefore, we have applied only PHALS to factorize the real datasets. For each of the datasets, $n_{\text{init}} = 10$ random initializations are used and the best solution in terms of data reconstruction error is selected. The maximum allowed number of iterations is set to $K_{\text{PHALS}} = 2000$ and a convergence criterion based on $\Delta_k$ is used as an additional stopping criterion.

*US Congressional voting dataset.* We first apply PHALS to a dataset containing 16 key votes of the United States congress for the year 1984 [34][5]. The votes of 435 representatives are coded by binary values: '1' for a vote in favor of the proposed bill and '0' for a vote against it. Missing votes in a given bill have been replaced by the corresponding majority vote. This allows to fully encode the dataset into a binary matrix of size $435 \times 16$. The dataset also contains information on the party of each representative (democrat or republican). Since there are 2 parties, PHALS is applied to the dataset with $R = 2$. The dataset plot and an illustration of the results are given in Fig. 3 (a-d). One can clearly observe that the patterns related to each component have almost disjoint support, indicating an opposing voting pattern for each component. The error rate on the reconstructed data using the retrieved BMF model is of 20%. By comparing the grouping of the representatives encoded by matrix $\hat{\mathbf{A}}$ with the information on the parties of each candidate, we found that PHALS can predict the party of the representative with an accuracy of 77%.

*Zoo dataset.* We also applied PHALS to a dataset containing the information on the presence or absence of a given feature, for example *hair*, *feathers*, *milk*, for different animals. The dataset [34][6] contains 15 binary features and an integer feature with the number of legs. These features are given for 101 animals. The animals in the dataset are categorized in 7 classes: mammals, birds, reptiles,

[5] `https://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records`
[6] `https://archive.ics.uci.edu/ml/datasets/Zoo`

fishes, amphibians, insects and a class containing many different invertebrate animals (*e.g. crab*, *worm*, *octopus*). We have encoded the integer variable corresponding to the number of legs using one-hot encoding. We apply PHALS to the resulting $101 \times 19$ binary matrix to try to group the animals by looking at the patterns given by the columns of $\hat{\mathbf{A}}$. The algorithm is applied with $R$ in the range 2 to 7 and the data reconstruction error for these values of $R$ are respectively 0.170, 0.116, 0.0928, 0.0771, 0.0693 and 0.0620. One can clearly see that beyond $R = 3$ the improvement on data reconstruction obtained by increasing $R$ is mild. This result seems to be similar to what has been presented in [23] for the analysis of the same dataset with PNL-PF.

The dataset and an illustration of the results for $R = 3$ are given in Fig. 3 (f-i). To simplify the interpretation of the results, the rows of the matrices, which correspond to different animals, have been reordered to correspond to continuous blocks of animals of the same category. Reordering has been carried out using the same order of the classes mentioned above, thus the first block of animals correspond to mammals, the second to birds, *etc.* One can observe that component 1 clearly corresponds to a continuous block of animals, in these case mammals. The second component mostly group together birds with two exceptions, *fruitbat* and *vampire*, which are also present in the group of mammals. The third group contains mostly fishes, but also some mammals (*e.g. dolphin*) and birds (*e.g. penguin*). Many insects and animals from the last category of invertebrate animals are not contained in any components. As the number of components is increased to $R = 7$, it has been observed that the retrieved BMF is not able to clearly separate the 7 underlying categories.

*Paleontological dataset.* Following closely [15, 40], we analyze data containing information on the localization of fossil mammals [35]. The objective is to apply PHALS to factorize a binary data matrix where the rows correspond to different genera of fossil mammals and the columns correspond to the different localities where they have been found. The data obtained from [35] is preprocessed in a similar manner as in [15, 40]. Fossils of small mammals are excluded from the dataset and only those retrieved in Europe are considered. We also removed genera which are too infrequent (less than 10 occurrences) and localities where only 1 genera has been found. As a result of this preprocessing, a binary matrix of size 254 (genera) $\times$1375 (localities) is obtained, where a '1' stands for the occurrence of at least one fossil of a given genus in a given locality and a '0' for its absence. We have applied PHALS to this dataset to see if the resulting BMF allows to find communities of mammals that appear in similar localities. The algorithm has been applied with $R$ in the range $2 - 7$. The algorithm seems to suffer from convergence issues for $R > 4$, generating factor matrices with spurious empty columns. To validate the results for $R \leqslant 4$, we have followed [40] and plotted

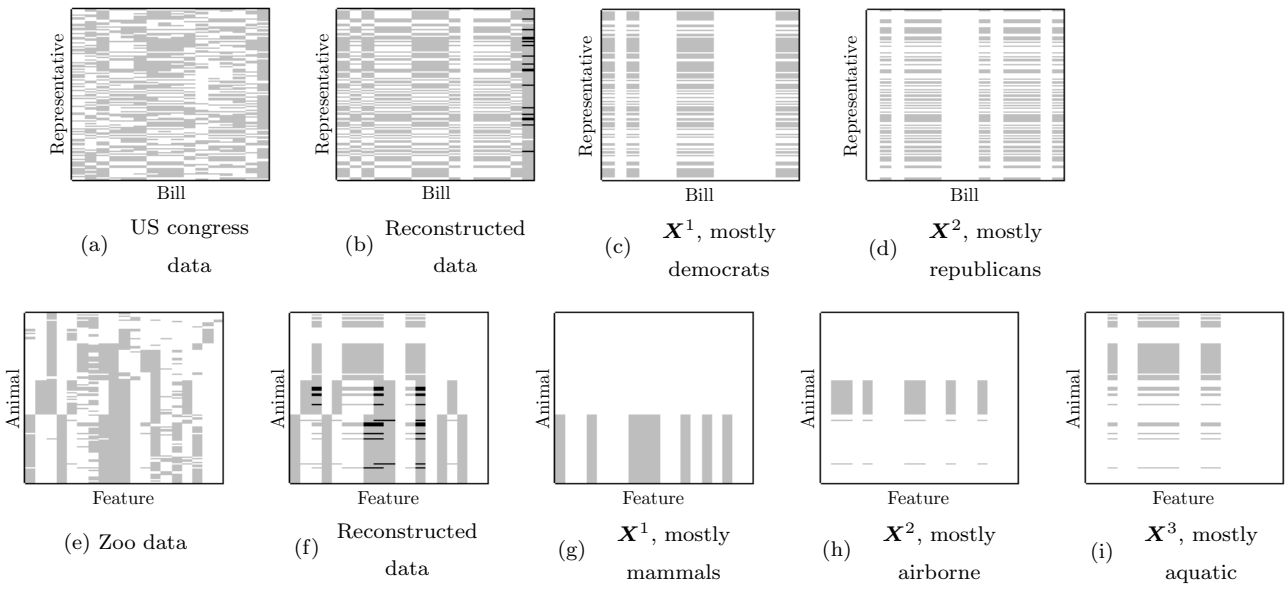



Figure 3: Real datasets and the results obtained with PHALS. In (a) and (e) the US congress voting dataset and the Zoo dataset are displayed. The gray color corresponds to a '1' in the underlying dataset matrix, while white color corresponds to '0'. In (b) and (f), the reconstructed data using the BMF model are displayed. The black color indicates intersections between BMF components' supports. The number of BMF components are $R = 2$ and $R = 3$ respectively. The rank-1 components $\boldsymbol{X}^r$ retrieved with PHALS are displayed on the right of the reconstructed data in (c), (d), (g), (h) and (i).

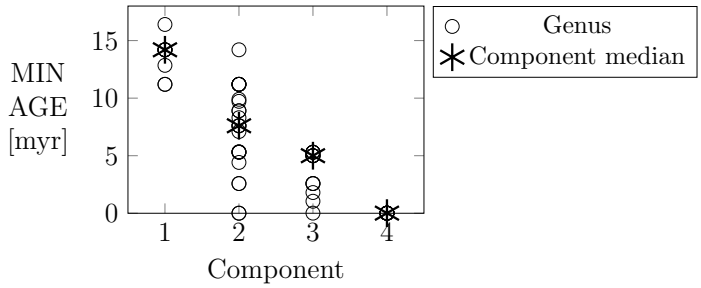



Figure 4: Minimum age in millions of years [myr] related to the genera of the different groups obtained by applying PHALS with $R = 4$ to the paleontology dataset [35].

the values of a variable related to the minimum age in millions of years of the localities where the different genera have been found. We have observed that as $R$ increases PHALS finds groups of fossils with increasing minimum age. The ages of the genera in the different groups for $R = 4$ are displayed in Fig. 4. The genera in each of the components of this figure are the following:

- Component 1: *Amphiperatherium*, *Amphitragulus*, *Andegameryx*, *Brachyodus*, *Cainotherium*, *Cynelos*, *Diaceratherium*, *Palaeogale*, *Protaceratherium*.
- Component 2: *Amphicyon*, *Anchitherium*, *Anisodon*, *Aureliachoerus*, *Brachypotherium*, *Bunolistriodon*, *Dicrocerus*, *Dorcatherium*, *Gomphotherium*, *Hemicyon*, *Hyotherium*, *Lagomeryx*, *Lartetotherium*, *Listriodon*, *Martes*, *Micromeryx*, *Palaeomeryx*, *Plesiaceratherium*, *Procervulus*, *Prodeinotherium*, *Prosantorhinus*, *Pseudaelurus*, *Styriofelis*, *Taucanamo*.
- Component 3: *Adcrocuta*, *Choerolophodon*, *Cremohipparion*, *Deinotherium*, *Dihoplus*, *Gazella*, *Helladotherium*, *Hipparion*, *Hippopotamodon*, *Hippotherium*, *Hyaenictitherium*, *Miotragocerus*, *Palaeotragus*, *Pliodiceros*, *Tragoportax*.
- Component 4: *Bison*, *Canis*, *Cervus*, *Equus*, *Lynx*, *Mammuthus*, *Panthera*, *Stephanorhinus*, *Sus*, *Ursus*, *Vulpes*.

By inspecting the median minimum ages of these groups, it seems that PHALS is able to retrieve animal communities that have lived in different ages.

*UN voting dataset.* As a last example of application, we analyze the grouping of countries produced by PHALS when used to factorize a binary matrix generated from the UN voting dataset [36][7]. We

[7] `https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/LEJUQZ`

follow a similar setting as considered in [15] and we encode in a binary matrix the votes during the cold-war period ($1946-1990$) of different countries for different UN roll-calls. We have removed from the dataset all roll-calls whose number of unknown votes is larger than 98 (half of the listed countries) and also all the roll-calls with unanimous results. The unknown votes in the remaining roll-calls have been replaced by the majority vote. For this dataset, encoding with '1' votes in favor of a roll-call leads to a very dense data matrix, whose BMF is difficult to retrieve and interpret. Therefore, to have a more sparse data matrix, we have encoded with '1' votes against a roll-call and with '0', votes in favor of it. PHALS has been applied to this dataset with $R$ in the range $2-7$. The data reconstruction error is respectively 0.0312, 0.0241, 0.0215, 0.0194, 0.0173 and 0.0158. Although a large part of the approximation improvement is observed when increasing $R$ from 2 to 3, when we analyze the groups of countries produced for each $R$, interesting results seem to appear up to $R = 6$.

When $R = 2$, we can find a component containing the following countries: Australia, Belgium, Canada, Denmark, France, West Germany, Iceland, Israel, Italy, Japan, Luxembourg, Netherlands, New Zealand, Norway, Portugal, UK, US. The second component contains 173 countries from different continents. If $R$ is increased to 3, the first 2 components are similar to those obtained with $R = 2$ and a third component groups countries from the socialist block: Belarus, Bulgaria, Cuba, Czechoslovakia, East Germany, Hungary, Mongolia, Poland, Russia, Ukraine. When increasing $R$ to 4, similar results are obtained and a component with US and Israel appears. While for $R = 5$, 2 components with countries from the occidental block of countries are produced. For $R = 6$, the component containing a large number of countries seems to contain much less countries than for smaller $R$ and a sixth component containing 36 countries from different continents appear. This last component gathers countries from the previously obtained component with a large number of countries but also countries from the occidental block (*e.g.* France) and from the socialist block (*e.g.* Cuba). Finally, for $R = 7$, the algorithm start finding components containing single countries (*e.g.* a component with only US).

### *4.8. Results for $XOR-2$ and $MAJ-3$*

The last experimental results concern the application of the GD and PHALS to a factorization setting different from BMF. We consider two other Boolean combining functions, the logical 'XOR' with two inputs $x_1 \oplus x_2$ ($\mathrm{XOR}-2$) and 3-term majority $\mathbb{1}_{(\sum_i x_i) \geqslant 2}(x_1, x_2, x_3)$ ($\mathrm{MAJ}-3$). Since the uniqueness properties of these factorizations are still very little understood, we only focus on testing the methods for data denoising. We consider a simulation setting similar to the one presented in Subsection 4.2, the main differences are that the underlying (clean) data are generated with the

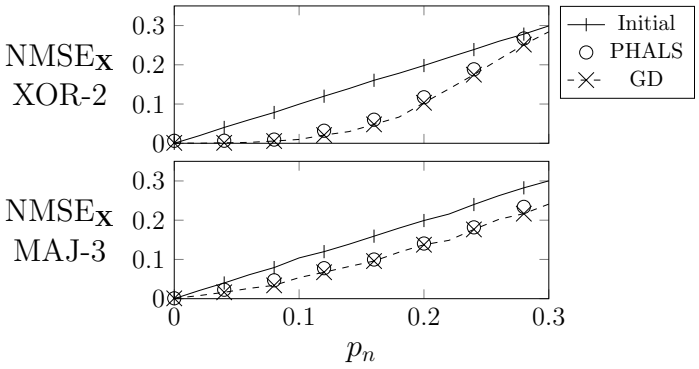



Figure 5: NMSE for the prediction of $\boldsymbol{X}$ of size $20 \times 20$ for generalized Boolean factorizations using the XOR $-$ 2 and MAJ $-$ 3 component combining functions. The decomposition algorithms are executed on a noisy version $\boldsymbol{Y}$ of $\boldsymbol{X}$. The probability $p_n$ of the binary noise that flips the elements of $\boldsymbol{X}$ is varied from 0 to 0.3 with increments of 0.02.

XOR and MAJ-3 functions, the maximum allowed number of iterations of the algorithms is set to $n_{\text{init}} = 5000$ and that an additional stopping criterion based $\Delta_k$ is used for ending the iterations. The results for the NMSE of data reconstruction are displayed in Fig. 5. One can observe that the algorithms denoise the data, since their NMSE is smaller than the NMSE for the noisy data (curve named *Initial* in the plot). In both cases, GD seems to achieve a slightly superior denoising performance than PHALS. It is also possible to observe that the factorizations do not seem to have the same robustness behavior against noise. The denoising performance for F2MF (XOR-2) for small noise intensity seems much superior than for the MAJ-3 factorization. This is intuitively expected, since the MAJ-3 factorization requires the estimation of more parameters for the same amount of data. For large noise intensities ($p_n \approx 0.3$), the opposite behavior is observed, with the MAJ-3 factorization leading to a superior denoising performance.

## 5. Conclusions and further work

In this paper, we have introduced a generalized framework for the Boolean factorization of binary matrices, where the "sum" between the rank-1 binary terms can be an arbitrary Boolean function. We proposed two iterative algorithms for achieving this factorization, based on gradient descent (GD) and on projected hierarchical alternating least squares (PHALS) approaches, respectively. Implementation details for the algorithms have been presented for BMF and compared through numerical experiments with state-of-the art algorithms from the literature.

From the results of the numerical experiments, it seems that the best performing algorithm is

PHALS, both in terms of performance of retrieving the factorization and of overall computation time. Although GD gives good results in terms of approximate factorization performance, its high complexity impedes its practical use on large datasets.

We have also tested PHALS to retrieve the BMF of real datasets. The components obtained seem to agree with those obtained in other works of the literature and with intuition on what would be possible groupings of the data. In this paper, we have not focused on the choice of the number of components $R$ and in the presentation of the results for the real datasets, we have chosen a value of $R$ that seemed to give stable results with components agreeing with intuition on the dataset. In practice, if no intuition on the expected components is available, a quantitative criterion for choosing $R$ may be used. Such criteria will be studied and tested in future work.

At the end of the experimental section, we have also presented results of applying PHALS and GD in a more general factorization setting where XOR $-$ 2 and MAJ $-$ 3 component combining functions are considered instead of the logical OR of BMF. Such matrices factorizations are not identifiable in general and thus may not be useful in data analysis. In future work, we would like to extend our general approach to the higher-order tensor setting and to verify whether the extended models are identifiable.

## References


[1] T. Li, A general model for clustering binary data, in: Proceedings of the eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, 2005, pp. 188–197.

[2] L. Kozma, A. Ilin, T. Raiko, Binary principal component analysis in the Netflix collaborative filtering task, in: 2009 IEEE International Workshop on Machine Learning for Signal Processing, IEEE, 2009, pp. 1–6.

[3] E. Nenova, D. I. Ignatov, A. V. Konstantinov, An FCA-based boolean matrix factorisation for collaborative filtering, in: FCAIR 2012 Formal Concept Analysis Meets Information Retrieval Workshop co-located with the 35th European Conference on Information Retrieval (ECIR 2013) March 24, 2013, Moscow, Russia, 2013, p. 57.

[4] M. Diop, S. Miron, A. Larue, D. Brie, Binary matrix factorization applied to Netflix dataset analysis, IFAC-PapersOnLine 52 (24) (2019) 13–17.

[5] E. Meeds, Z. Ghahramani, R. M. Neal, S. T. Roweis, Modeling dyadic data with binary latent factors, Advances in neural information processing systems 19 (2007) 977.

[6] Z.-Y. Zhang, T. Li, C. Ding, X.-W. Ren, X.-S. Zhang, Binary matrix factorization for analyzing gene expression data, Data Mining and Knowledge Discovery 20 (1) (2010) 28–52.

[7] S. Tu, L. Xu, R. Chen, A binary matrix factorization algorithm for protein complex prediction, in: 2010 IEEE International Conference on Bioinformatics and Biomedicine Workshops (BIBMW), IEEE, 2010, pp. 113–118.

[8] H. Lu, J. Vaidya, V. Atluri, Optimal Boolean matrix decomposition: Application to role engineering, in: 2008 IEEE 24th International Conference on Data Engineering, IEEE, 2008, pp. 297–306.

[9] S. Talwar, M. Viberg, A. Paulraj, Blind separation of synchronous co-channel digital signals using an antenna array. I. algorithms, IEEE Transactions on Signal Processing 44 (5) (1996) 1184–1197.

[10] A.-J. Van der Veen, Analytical method for blind binary signal separation, IEEE Transactions on Signal Processing 45 (4) (1997) 1078–1082.

[11] A. I. Schein, L. K. Saul, L. H. Ungar, A generalized linear model for principal component analysis of binary data, in: International Workshop on Artificial Intelligence and Statistics, PMLR, 2003, pp. 240–247.

[12] J. De Leeuw, Principal component analysis of binary data by iterated singular value decomposition, Computational statistics & data analysis 50 (1) (2006) 21–39.

[13] S. Lee, J. Z. Huang, J. Hu, Sparse logistic principal components analysis for binary data, The annals of applied statistics 4 (3) (2010) 1579.

[14] Z. Kang, C. J. Spanos, Sequential logistic principal component analysis (SLPCA): Dimensional reduction in streaming multivariate binary-state system, in: 2014 13th International Conference on Machine Learning and Applications, IEEE, 2014, pp. 171–177.

[15] A. Lumbreras, L. Filstroff, C. Févotte, Bayesian mean-parameterized nonnegative binary matrix factorization, Data Mining and Knowledge Discovery 34 (6) (2020) 1898–1935.

[16] Z. Zhang, T. Li, C. Ding, X. Zhang, Binary matrix factorization with applications, in: Seventh IEEE International Conference on Data Mining (ICDM 2007), IEEE, 2007, pp. 391–400.

[17] N. Gillis, S. A. Vavasis, On the complexity of robust PCA and $\ell$1-norm low-rank matrix approximation, Mathematics of Operations Research 43 (4) (2018) 1072–1084.

[18] P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, H. Mannila, The discrete basis problem, IEEE transactions on knowledge and data engineering 20 (10) (2008) 1348–1362.

[19] R. Belohlavek, V. Vychodil, Discovery of optimal factors in binary data via a novel method of matrix decomposition, Journal of Computer and System Sciences 76 (1) (2010) 3–20.

[20] M. Trnecka, R. Vyjidacek, Revisiting the Grecon algorithm for Boolean matrix factorization, Knowledge-Based Systems (2022) 108895.

[21] T. Makhalova, M. Trnecka, From-below Boolean matrix factorization algorithm based on MDL, Advances in Data Analysis and Classification 15 (1) (2021) 37–56.

[22] D. D. Lee, H. S. Seung, Learning the parts of objects by non-negative matrix factorization, Nature 401 (6755) (1999) 788–791.

[23] S. Miron, M. Diop, A. Larue, E. Robin, D. Brie, Boolean decomposition of binary matrices using a post-nonlinear mixture approach, Signal Processing 178 (2021) 107809.

[24] D. DeSantis, E. Skau, D. P. Truong, B. Alexandrov, Factorization of binary matrices: Rank relations, uniqueness and model selection of Boolean decomposition, ACM Transactions on Knowledge Discovery from Data (TKDD) (2020).

[25] H. Nguyen, R. Zheng, Binary independent component analysis with or mixtures, IEEE Transactions on Signal Processing 59 (7) (2011) 3168–3181.

[26] S. Ravanbakhsh, B. Póczos, R. Greiner, Boolean matrix factorization and noisy completion via message passing, in: International Conference on Machine Learning, PMLR, 2016, pp. 945–954.

[27] A. Cichocki, A.-H. Phan, Fast local algorithms for large scale nonnegative matrix and tensor factorizations, IEICE transactions on fundamentals of electronics, communications and computer sciences 92 (3) (2009) 708–721.

[28] D. DeSantis, E. Skau, B. Alexandrov, Factorizations of binary matrices–rank relations and the uniqueness of Boolean decompositions, arXiv preprint arXiv:2012.10496 (2020).

[29] J. E. Cohen, U. G. Rothblum, Nonnegative ranks, decompositions, and factorizations of nonnegative matrices, Linear Algebra and its Applications 190 (1993) 149–168.

[30] K. H. Kim, Boolean matrix theory and applications, Vol. 70, Dekker, 1982.

[31] V. L. Watts, Boolean rank of Kronecker products, Linear Algebra and its Applications 336 (1-3) (2001) 261–264.

[32] T. Watson, Nonnegative rank vs. binary rank, arXiv preprint arXiv:1603.07779 (2016).

[33] Y. Crama, P. L. Hammer, Boolean functions: Theory, algorithms, and applications, Cambridge University Press, 2011.

[34] D. Dua, C. Graff, UCI machine learning repository (2017).
URL `http://archive.ics.uci.edu/ml`

[35] The now community. new and old worlds database of fossil mammals (now)., `https://nowdatabase.org/now/database/`, accessed: 2022-06-15.

[36] E. Voeten, Data and analyses of voting in the un general assembly, in: B. Reinalda (Ed.), Data and Analyses of Voting in the UN General Assembly, Routledge Londres, 2013.

[37] J. Nocedal, S. Wright, Numerical optimization, Springer Science & Business Media, 2006.

[38] P.-A. Absil, R. Mahony, B. Andrews, Convergence of the iterates of descent methods for analytic cost functions, SIAM Journal on Optimization 16 (2) (2005) 531–547.

[39] Y. Xu, W. Yin, A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion, SIAM Journal on imaging sciences 6 (3) (2013) 1758–1789.

[40] E. Bingham, A. Kabán, M. Fortelius, The aspect Bernoulli model: multiple causes of presences and absences, Pattern Analysis and Applications 12 (1) (2009) 55–78.